# REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

| | | |
|---|---|---|
| 1. AGENCY USE ONLY ( Leave Blank) | 2. REPORT DATE<br>June 13, 2003 | 3. REPORT TYPE AND DATES COVERED<br>Final Report, ~~2002-2003~~ 01 Sep 02 28 Feb 03 |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>Advanced Broadband Intrusion Detection Engine (ABIDE): Report on Seedling Project | 5. FUNDING NUMBERS<br>DAAD19-02-1-0404 |
| 6. AUTHOR(S)<br>Jonathan Smith, Michael Greenwald, E Lewis, Honghui Lu | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>University of Pennsylvania, Dept. of Comp. & Info Science, 3330 Walnut St., Philadelphia, PA 19104 | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br>U. S. Army Research Office<br>P.O. Box 12211<br>Research Triangle Park, NC 27709-2211 | 10. SPONSORING / MONITORING AGENCY REPORT NUMBER<br>~~ARO~~<br>44233.1-CI |

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

| | |
|---|---|
| 12 a. DISTRIBUTION / AVAILABILITY STATEMENT<br>Approved for public release; distribution unlimited. | 12 b. DISTRIBUTION CODE |

13. ABSTRACT (Maximum 200 words)

ABIDE, the "Advanced Broadband Intrusion Detection Engine", is a model for applying parallel processing to the increasing bandwidths present in optical fibers in a manner which will scale with increases in the number of lambdas in a WDM scheme. Our initial support from ARO was used to investigate design parameters, and we report a scheme that we believe will in fact allow sophisticated intrusion detection to operate on the entirety of a fiber's bandwidth. The design principle we employ is novel, consisting of alternating bands of filtering and aggregation functions organized into a virtual tree, which is then mapped to the underlying ABIDE hardware system. The aggregation/filtering adjacencies allow localized tuning at the boundary. For example, if an upstream filtering system is overwhelmed, predecessor (downstream) aggregation functions must get backpressure to decrease the number of streams merged. We call this scheme Filtering Aggregation Bands (FAB).

We are prepared to continue this research and perform a more detailed experimental investigation for ARO along the lines of our original proposal.

| | |
|---|---|
| 14. SUBJECT TERMS | 15. NUMBER OF PAGES<br>12 |
| | 16. PRICE CODE |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFICATION OR REPORT<br>**UNCLASSIFIED** | 18. SECURITY CLASSIFICATION ON THIS PAGE<br>**UNCLASSIFIED** | 19. SECURITY CLASSIFICATION OF ABSTRACT<br>**UNCLASSIFIED** | 20. LIMITATION OF ABSTRACT<br>**UL** |

NSN 7540-01-280-5500

**Standard Form 298 (Rev.2-89)**
Prescribed by ANSI Std. 239-18
298-102

Enclosure 1

# Advanced Broadband Intrusion Detection Engine (ABIDE) Report on Seedling Project

*abide* (def.: to wait patiently for; to endure)

Jonathan Smith, Michael Greenwald, E.C. Lewis, and Honghui Lu

June 13, 2003

**Abstract**

Executive Summary

ABIDE, the "Advanced Broadband Intrusion Detection Engine", is a model for applying parallel processing to the increasing bandwidths present in optical fibers in a manner which will scale with increases in the number of lambdas in a WDM scheme. Our initial support from ARO was used to investigate design parameters, and we report a scheme that we believe will in fact allow sophisticated intrusion detection to operate on the entirety of a fiber's bandwidth. The design principle we employ is novel, consisting of alternating bands of filtering and aggregation functions organized into a virtual tree, which is then mapped to the underlying ABIDE hardware system. The aggregation/filtering adjacencies allow localized tuning at the boundary. For example, if an upstream filtering system is overwhelmed, predecessor (downstream) aggregation functions must get backpressure to decrease the number of streams merged. We call this scheme Filtering Aggregation Bands (FAB).

We are prepared to continue this research and perform a more detailed experimental investigation for ARO along the lines of our original proposal.

## 1 Introduction

It is commonly (and incorrectly) believed that intrusion detection cannot be performed in *the core* of modern broadband networks. Our long term goal, the construction of the network-embedded Advanced Broadband Intrusion Detection Engine (ABIDE) will demonstrate the capability for network-embedded programmability, specifically for intrusion detection and monitoring for wavelength-division multiplexed (WDM) [1] broadband optical networks. This ultimate version of ABIDE would use network processors such as the Intel IXP1200 or IXP2800 interconnected in a tree to achieve the highest performance. The proposed architecture however, raises many technical questions, as it relies on a complex synthesis of advanced hardware and novel software.

We set out to investigate a key technical question in building a scalable broadband intrusion detection engine, namely, how to manage the concurrent collection of data from multiple sensing points. Our initial approach was to study this question by building a small prototype system in software, using conventional cluster-based distributed computing. Our team's earlier experiences building a distributed firewall suggest that scalable security systems can be constructed by careful localization of decision making. In intrusion detection, we believe that a combination of distributed sensors under control of a common policy, and an intelligent fusion of data from these sensor systems, can lead to scalable intrusion detection systems.

This document reports on our preliminary work to address this question, in a project launched using seedling funds from the ARO. The most basic observation that arose from our analysis was that, while we still believe that a scalable IDS engine is achievable, a single, static, configuration must either be over-provisioned (and expensive) or else provide incomplete coverage. Therefore, we need a more sophisticated architecture than we first envisioned. In particular, we have (i) identified several new fundamental research questions related to using parallel computing for scalable network surveillance, and (ii) proposed a new architecture (FAB: Filtering Aggregation Bands) to address these issues.

We believe that the new issues we raise will affect all such parallel efforts to achieve scalable network surveillance, and will require any such effort to attack the problem of dynamic reconfigurability.

The rest of this document reviews the problem space we are investigating, reviews our initial proposal, summarizes both the new technical questions we have uncovered and our proposed architecture to address them, and briefly discusses other comparable research.

## 2 Broadband network challenges — computing, communications and performance

In this section, we outline the challenges of broadband intrusion detection. First is the raw bandwidth of the networks, and second is the nature of the intrusion detection task itself.

### 2.1 The Challenge of Fiber Optic Communications

The merging of computing and communication [3] has had positive consequences for both fields, with networks enhancing the capability of computing systems and vice versa. Significant leverage has been gained from the common use of electronics in both domains, as well as the ability to deploy processing coupled to the networking devices in advanced networked systems. For example, the entire "World Wide Web" architecture is predicated upon sets of services accessible via URLs; these services are provided by services co- located with large-scale network infrastructures provided by ISPs.

The many advantages of fiber-optic systems have made them the transport medium of choice for advanced communications systems, such as those deployed at the core of backbone networks [1]. Transparent all-optical systems achieve the greatest advantage from the technology, particularly in the form of wavelength-division multiplexed transport systems [5]. A major challenge to system architects is the coupling of processing to optical systems with total throughputs *(bandwidth*wavelengths)* in the terabit/second regime. With limited bus bandwidths (*e.g.*, PCI at *ca.* 2Gbps) and DRAM bandwidths (*ca.* 1Gbps on P6, but larger on many other architectures), conventional approaches to constructing an Optical Supervisory Channel (OSC) processor will result in limited performance and functionality relative to the capabilities of the transport system.

A *novel* architectural solution to matching core network bandwidths to processing for tasks such as intrusion detection (discussed in the next section) is needed. Since we are interested in WDM networks, we must multiply the serial throughputs by factors currently ranging from 16 to 80 or more. With many advanced IP services and active networking [4], there is a strong pressure for *increased* performance and flexibility in an OSC processor. We believe that novel architectural ideas can help us crack the boundaries in the network and processor architecture communities.

### 2.2 What is Intrusion Detection and why is it hard to scale?

Intrusion detection is based on pattern matching of packets, either singly or in series. An excellent short reference is Paxson's paper on the *Bro* intrusion detection system [8]. A more comprehensive introduction to work done before 2000 on Intrusion Detection systems is available from the CMU Software Engineering Institute [13]. The task performed by an intrusion detection system (IDS) is to search for specified sets of patterns on a monitored link. These systems are deployed on broadcast LANs or as "bumps" in a path on which data are to be monitored. When an IDS detects a pattern associated with an intrusion (an attack signature), it generates an alert message which is sent to a specified destination, either local or remote.

While firewalls tend to limit their interest to packet headers, so that firewall rules incur a per-packet cost proportional to the cost of a rule, intrusion detection systems look at both packet headers and packet bodies. This means three challenges ensue. First, the complexity of the patterns being searched for imposes a major computational burden on the processing capacity of the intrusion detection machine. Second, since the intrusion detection system looks at packet bodies as well as packet headers, the per-packet cost is a function of the size of the packet as well as the complexity of the rules. Third, since the bandwidth of the network infrastructures is increasing, this means that the real performance challenge is a *bandwidth * rule* complexity product. This is particularly true when the network infrastructure is DWDM, as the total bandwidth, as we have discussed above, is a function of the number of wavelengths and the capacity of each wavelength.

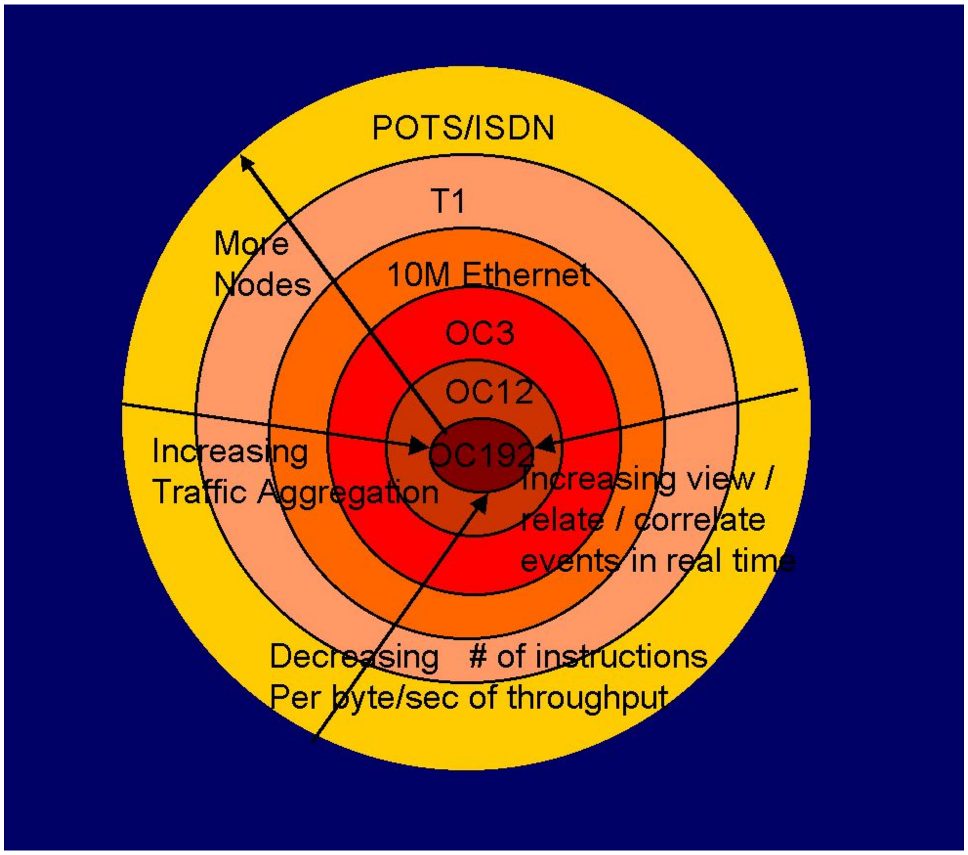
POTS/ISDN
T1
10M Ethernet
OC3
OC12
OC192
More Nodes
Increasing Traffic Aggregation
Increasing view / relate / correlate events in real time
Decreasing # of instructions Per byte/sec of throughput

Figure 1: Tradeoffs in Network Intrusion Detection Systems

### 2.3 This problem exceeds the capacity of a single computer

The bandwidths and processing capacity available from even the fastest modern processors cannot keep up with the network infrastructure. Essentially, what would be required is a capacity to perform string matching on each wavelength at its line rate (these are now ca. 10 billion bits per second, a capacity beyond the memory performance of workstations).

Thus we must look at parallel solutions. The key questions will be the data gathering and fusion architecture. In the next section, we examine a solution to ABIDE using special purpose processors. This will set the stage for the research questions we have begun to address with our initial investigation and which we hope to pursue in our prototype ABIDE system, based on interconnection patterns overlaid on a cluster of conventional computers. We believe this effort will tell us if a full-scale ABIDE is possible, and where its use will be sensible.

## 3 Approach

### 3.1 The ABIDE Approach using special purpose network processors

ABIDE couples active networking[4], an approach to programmable network elements, to high-performance parallel computing. Programmability allows insertion of intrusion detection algorithms (developed by others, and adapted to ABIDE) deep into the network core. The basic principle is use of add-drop demultiplexers (ADMs) with outputs routed into a tree-structured single-instruction multiple data (SIMD) architecture parallel processor. The processor is structured as a tree, with Intel IXP1200 (eventually IXP2800) network processors near the leaf nodes, feeding more general-purpose machines higher up in the tree. ABIDE exploits

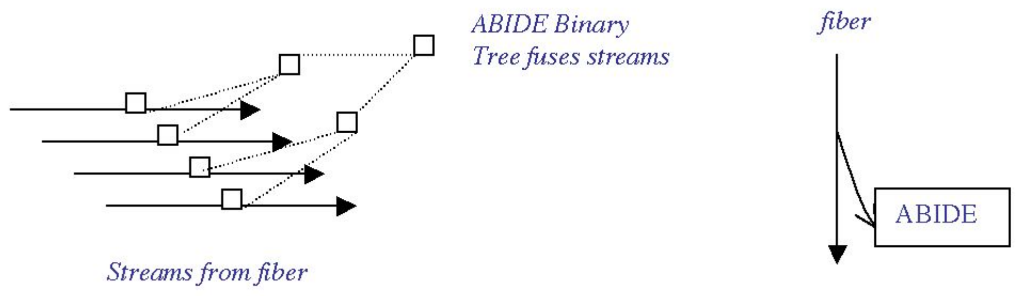

Figure 2: ABIDE Architecture and Deployment

the large body of federally-supported work in massively parallel machines (such as DADO[7] and Non-VON) in the 1980s, to provide functions explored in the 1990s in active networking research, to solve critical infrastructure problems for the 2000s and beyond.

## 3.2 The ABIDE System Architecture

Parallel architectures. such as the binary tree (each processor has two children, which recursively have children) are highly scalable and can easily be coupled to a WDM system.

ABIDE processing elements would be deployed adjacent to a wavelength selective optical crossconnect (OXC) node[1] supporting an Optical Supervisory Channel (OSC). Demultiplexed streams from the fiber are delivered to ABIDE. ABIDE processes all (or selected channels) from the fiber. Any wavelength is a candidate for processing, although ABIDE will typically designate selected channels (the focus of intrusion detection or surveillance) for control and processing, while other channels pass through unprocessed. Selection is performed with a wavelength add-drop multiplexor / wavelength converter.

Intrusion detection and surveillance applications of ABIDE will take advantage of both network-embedded processing and the tree machine's "impedance-matching" between processing performance and network capacity. Sorting, searching and selection have probably consumed more processor cycles than any other application. ABIDE would allow line rate comparisons and selection of bitstreams from one or more wavelengths (depending on ABIDE performance and the application processing burden of the task).

Leaf processors in the tree frame process packets taken from the network. The packet format demarcates which bits are used for control and which for transport. This structure is used, e.g., in the IP architecture, which makes the packet header the locus of control, so that header manipulations can be localized, and result in per-packet overheads that do not induce large per-byte overheads). Control operations, since they are localized, can take longer as they are shared across the larger number of data bytes in the packet. Intrusion detection, as we saw above, must look at bodies as well as headers, so it cannot take advantage of the packet structure to increase performance, as can routers and firewalls.

## 4 Research Questions in the ABIDE Architecture

ABIDE can make a large difference in the monitoring and control of the network infrastructure. There are many challenges in the entire ABIDE research agenda. What we see as hard problems requiring some initial investigation are the host interconnection strategy and the data fusion paradigm / programming model.

For the long term, Intel IXP1200 network processors would be ideal front ends. However, a set of Pentiums configured as a cluster can provide insight into how to interconnect a parallel intrusion detection or general-purpose surveillance system.

The cluster would consist of Intel processors interconnected by Gigabit Ethernet LAN technology. The interconnection technology would constrain data flow along any particular link. A logical edge would be

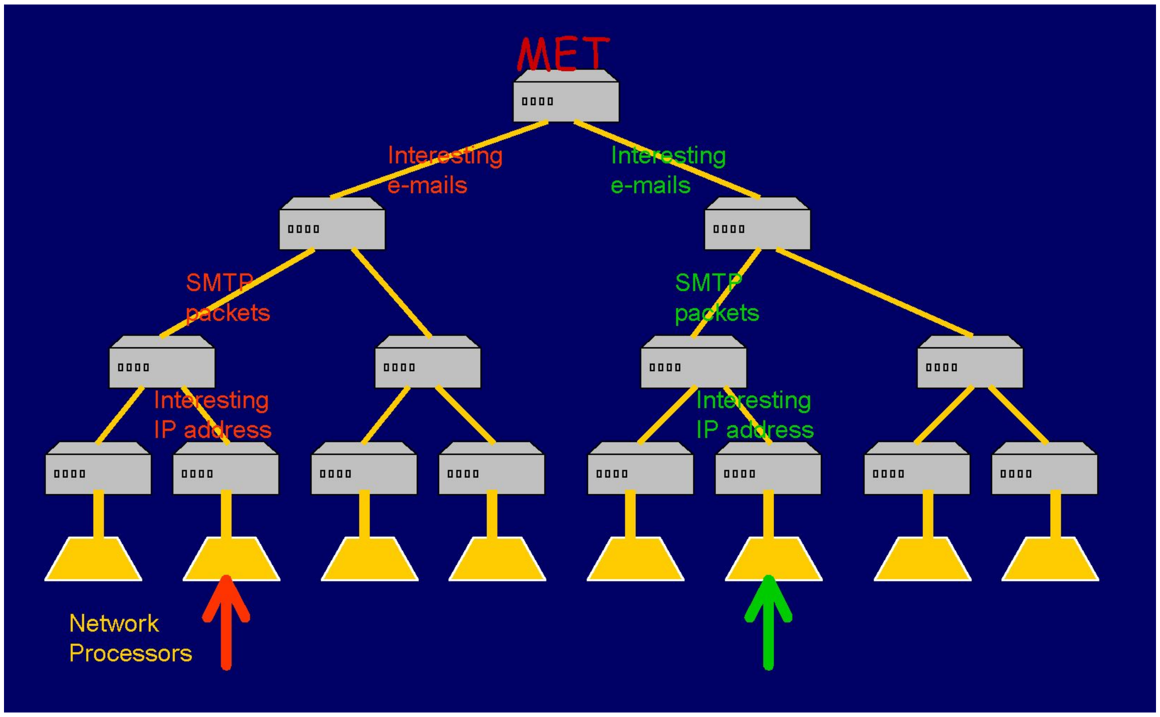

Figure 3: An idealized parallel optical scan engine

defined from a subset of cluster processors, and then traffic traces would be pushed against this edge, as if data was being derived from a multiplicity of wavelengths of a WDM fiber. The leaves will perform the base intrusion detection functions of pattern matching, and when they detect a pattern of interest, indicating a possible intrusion, they indicate an exception to the "parent" node in the topology. Our intention is to use either *Bro* or **snort** as the prototype intrusion detection system, operating on OpenBSD or Linux cluster nodes. The outputs from these systems will be forwarded "up" the tree structure towards the management station at the root of the tree.

**Snort**, for example, relies on the popular UNIX `pcap` packet capture library. The packet capture is used to feed packets to the **snort** packet processing logic. This logic begins with a rule chain of zero or more preprocessors which might edit or use state - such as the location in a higher level object such as a message - to prepare the packet for processing by one of the three rule trees that snort provides. The rule trees exist for dropping, passing and logging packets, which are the essential operations in an intrusion detection system. Real-time alerts can be sent via the syslog() call, or if it is in use, via the Samba system. Back-end tools can process the syslog entries and filter them for forwarding up the ABIDE tree. **Snort** rules take the form:

```
alert tcp any any -> 192.168.1.0/24 111 (content:"00 01 86 a5|";msg: "mountd access";)
```

These rules are powerful and have been used in many applications. Many details can be found at: `http://www.dpo.uab.edu/~andrewb/snort/snortdoc/snort.html`. Wenke Lee and his group at Georgia Tech have ported **Snort** rules to the Intel IXP processors [14], thus demonstrating feasibility[1], and allowing us to leverage off of existing technology.

The precise mapping of either **Snort** or *Bro* rules to ABIDE is still a matter of research. Two basic approaches seem feasible.

[1]That effort (AIDE) aims at achieving scalability by pushing processing to end-hosts and doing very little in the core of the network. ABIDE takes a diametrically opposed view, aiming to do very little at the end-host and doing most of the work at the core of the network, where more traffic is aggregated.

First, each leaf node can inspect a single packet. Assume that running a packet against the pattern matcher takes up to $K$ times as long as the transmission time for that packet. Then, a system with $K \times C$ leaf nodes should be able to monitor $C$ channels in real-time. We assume that most packets are benign and will not trigger alarms, so that we achieve reasonable data reduction as we go up the tree to the root(s).

Alternatively, assume that each leaf node is responsible for a single rule[2]. The system must comprise enough leaf nodes to cover every rule for every channel being monitored, and therefore the system is as scalable as the first approach.

Intrusion detection systems, such as **Snort**, deal with stateful filters (*e.g.* multi-packet patterns that indicate possible intrusion such as port-scanning). Under the packet-per- node architecture such patterns can be detected by keeping state at the leaf nodes, in a manner similar to the current architecture of **Snort**. Under the rule-per-node architecture, general purpose nodes higher in the tree must detect such situations by detecting that multiple primitive patterns have been detected in a short amount of time or in an interesting sequence.

We can learn a great deal about building a scalable parallel intrusion detection system from experiments done on a commodity rack of machines. First, we can estimate the degree to which the rules cause data reduction. Our belief is that significant data reductions almost always take place from the leaves to the first parent. A key question is whether the fusion occurring when the tree is more than a single level deep will actually cause significant expansion in the data throughput required as data move up the tree. If this expansion occurs, then a simplistic approach is impossible, as it will be limited by the bandwidth into the root. However, if data reduction can be achieved (e.g., by placing filtering rules or compression at the data fusion locations) then the system can be scaled with an appropriate number of processors.

The (expected) data reduction may turn out to be sensitive to the choice of packet-per- node or rule-per-node. A related question we can address experimentally is whether the degree of data reduction at each level of the tree is specifically related to the structure of **Snort** or *Bro* rules. One can conjecture that more effective data reduction comes at a higher computational cost. If so, perhaps the system will scale better with weaker data reduction at the leaves (reducing the number of leaf nodes needed to keep up with a channel), but with additional data reduction occurring higher in the tree (on cheaper commodity processors). In the extreme case, perhaps forms of rules that differ from **Snort** or *Bro* are more appropriate to our tree architecture — can we structure intrusion detection rules to balance the filtering load more evenly over different levels of the hierarchy?

This leads to the second experimental result. When we measure the absolute bandwidths each leaf node and internal node can handle, we can determine what depth and branching factor in the tree are suitable for this task (it may turn out that filtering effects are so dramatic that each internal node can handle 5-10 leaves, which is very attractive from a cost perspective).

A third experimental result, based on the traffic, and its variance with respect to data reductions from filtering, we can decide how much extra capacity is necessary in the system for performance guarantees. For example, users may want to guarantee that at most 10% of packets are lost, at most 0.1% of packets are lost, or that no packets are lost. With appropriate variance estimates we can appropriately overbuild the system to meet user expectations.

Finally, since our system will be a working distributed intrusion detection system, experiments regarding the use of the system with different network configurations can be performed. While the system was inspired from a task which suggests a Single-Instruction Multiple Data (SIMD) style of parallelism (where a single rule set is applied uniformly across a multiplicity of network links), that of monitoring a multiple wavelength fiber network, other uses are clearly possible without much creativity. For example, independent elements of the rule set could be distributed across a multiplicity of ABIDE nodes, allowing load balancing across systems with very computationally-intensive rule sets. Since the results are fused, the management station gets to exploit the parallelism in this fashion as well.

[2]In point of fact, each node can be responsible for as many rules as can be processed during the transmission time of a single packet. A single rule per leaf node is the worst case — if ABIDE can handle a single rule per node, then it can handle more realistic situations.

## 5 Preliminary Observations

(draft)

There are two basic goals of applying parallel processing to network surveillance for intrusion detection. First, we want to handle a large amount of data in a short amount of time — multiple processors may be able to handle loads that individual processors cannot handle. Second, we want the monitoring facility to scale easily with the growth of network bandwidth — a viable design should permit an increase in capacity by adding processors in proportion to the increase in monitored traffic per second.

If each packet could be treated completely independently then it would be trivial to implement a fully scalable parallel monitoring system. As noted earlier, if a single processor can handle a packet (check the packet against *all* rules) in $K$ times the transmission time of a single packet, then we can simultaneously monitor $C$ channels by using $C \times K$ processors, and let each processor handle only every $K$th packet on a given channel.

However, intrusion detection cannot treat each packet independently. First, many rules refer to recent network traffic history in order to decide whether an anomolous or dangerous event must be flagged. Second, many rules refer to messages or streams that are comprised of multiple packets. It is impossible, in general, to know which processor need handle a given packet until the packet has been classified. Therefore, in order to handle a given packet, the required state and the packet contents must somehow migrate towards each other.

This is true at all levels of abstraction — we cannot know which processor has the state to handle a given message (for example, it may belong to some higher level flow) until the packets have been combined into a message and the message classified.

One can choose to maintain consistent shared state across many of the processors, updating the shared state every time a relevant event occurs on any processor. Alternatively, one can keep state (other than classification state) local to a single processor (or a very small number of processors) and forward packets and events up the tree-like graph to get to the desired processor.

We believe that the latter architecture will be significantly more scalable than the former, reducing the costs of inter-processor communication. Our basic architectural aim is to spread the packet processing load across as many processors as possible, but breaking up or combining rules to keep a balanced load across all processors, and to localize state as much as possible.

There are topological and algorithmic questions we must answer, however, in order to make this architecture viable. The topological questions are essentially static: First, what is the *logical topology* that will maximize both load sharing and throughput. Second, what is the underlying *physical topology* that best supports reconfiguration of the logical topologies. The algorithmic questions are dynamic: First, for a given (changing) load, how do we choose the right virtual topology to embed in the physical topology? Second, the topology is chosen based on history; however, there are likely to be intervals where the load overwhelms the intrusion detection engine — how do we deal with overload?

### 5.1 Logical topology

We logically organize the processors into a network called the FAB fabric. The FAB fabric is a tree-like network of general purpose processors terminating at a root node that acts as a management and reporting station. We divide the nodes in the network into *bands*. Bands closer to the root deal with events at a higher level of abstraction than nodes in bands farther from the root. Bands farther from the root deal with larger quantities of (aggregate) raw data. Data enters the network through network processors directly connected to leaves in the network graph. Each band filters events from lower bands, aggregates the unfiltered events into more abstract (higher level) events, and passes the more abstract event to the appropriate parent in the next band.

The topology of the FAB, and the allocation of the number of processors per rule (or rules per processor) depends on the set of rules we are using, and the recent history of packet traffic.

We start at the lowest band, consisting of network processors directly extracting packets from the optical fiber. If a single processor can handle a packet (check the packet against *all* local, first level, rules) in $K$ times the transmission time of a single packet, then we can simultaneously monitor $C$ channels by using $C \times K$ processors, and let each processor handle only every $K$th packet on a given channel. But this is a

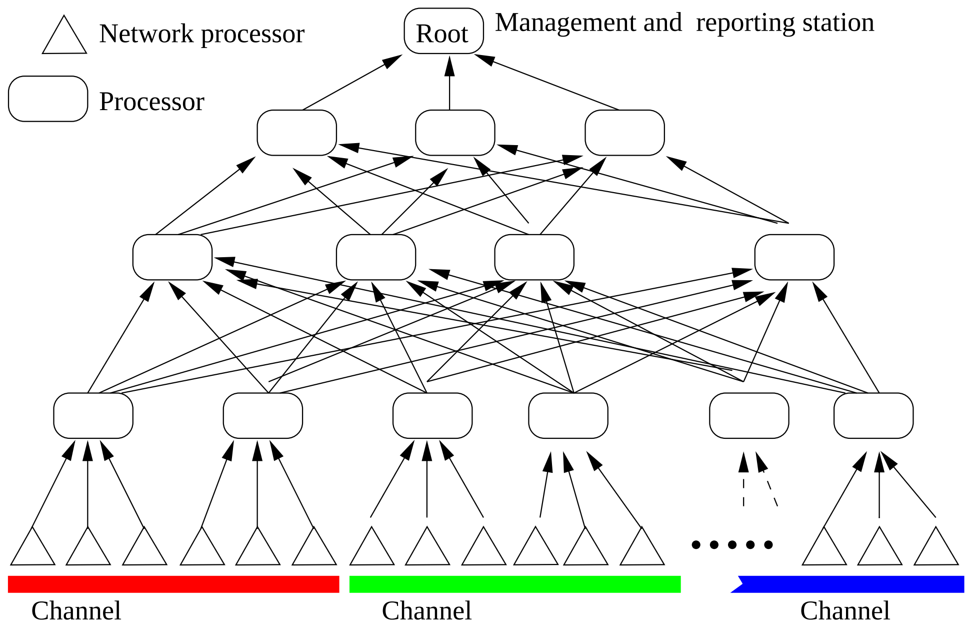

Figure 4: Virtual topology for FAB parallel optical scan engine, showing breakdown by channel into leaf nodes, and showing both fan-in (from data reduction) and fan-out (from aggregation) going up the tree.

minimum. We require that the next band in the FAB pipeline should be emptied before the packets from the leaves are passed up. This may require more than $C \times K$ leaf processors, if we need the per-band latency to be larger than $K$. Although the FAB in general will be reconfigurable, the leaves themselves are special purpose network processors, so we will probably have to overprovision the lowest (leaf) band.

Generally there are fewer nodes in band $N$ than in band $N+1$. The root node is at band 0. All nodes in band $N$ are *parents* of *every* child in band $N+1$. In other words, a node in band $N$ communicates directly with all nodes in band $N+1$ (its children) and all nodes in band $N-1$ (its parents), but with no other nodes.

Figure 4 shows the virtual topology of the FAB network.

It is a goal of the logical topology to ensure that, on average, the latency of every band should be equal to the latency of the band of leaf nodes. There is a competing goal, namely, that we have the fewest possible total number of bands in order to reduce the total latency. We monitor the behavior of the processors in order to dynamically reconfigure the virtual topology to reach these goals. The basic questions are where we break up each rules (what constitutes the band boundaries)? And, how wide (how many nodes) should each band be?

Assume a given, trial, topology. If the input queues on a given processor grow long, it means that processor is overloaded. The solution is straightforward: split the incoming data “stream” (in other words, change virtual topology by adding a node to the band). On the other hand, if the output queue grows long, it means that the communication channel overloaded (change physical embedding).

We will discuss the physical embedding in the next section, but for now, note that in general, packets/events are routed to the parent that has the appropriate state stored locally. So, although, every node in band $N$ is a parent of every node in band $N+1$, in practice certain pairs of nodes are more likely to communicate — a fact we can take advantage of when we embed the logical topology into our physical FAB fabric.

If band $N$ is already as wide as band $N+1$ and input queues are still overloaded, then the latency in band $N+1$ is too high, so the solution is to split it into two bands. Such a split is accomplished by reducing the number of rules in each processor in the band (and if only 1 rule, then splitting that rule into sub-rules). The embedding of the reconfiguration has an interesting property at this band boundary: we can then layout

Figure 5: Physical topology with virtual overlay.

this band with expected communication to only one parent.

At each band, we filter the incoming events. Those that trigger no rules are dropped. Those that trigger rules are aggregated into higher level events. Both filtering and aggregation can reduce the volume of data as we proceed through one more band towards the root.

## 5.2 Physical topology

The basic goal of the physical architecture is to support the dynamic reconfiguration of the communication paths between the nodes to support the desired virtual topology. In other words, we want to be able to embed the virtual topology in a given physical architecture. What physical topology is most suitable?

As discussed earlier, the desired virtual topology is a directed graph with a single root. The non-root nodes are formed into bands, such that every node in band $N$ is a child of every node in band $N+!$. Therefore, no connection is needed between nodes in bands that are more than one apart. Similarly, peers (nodes in the same band) don't need to communicate. However, the need to communicate with all parents and all children, may sometimes require communication with peers in order to orward/pass-through data to reach a specific parent that is not directly connected.

To implement this virtual topology we choose to embed it into a specific, extensible, physical topology. The physical FAB fabric consists of nodes connected in clusters. Each general purpose processor is connected to a gigabet ethernet control plane that serves solely as a channel to control reconfiguration. In addition, each processor belongs to two clusters. Each cluster consists of nodes allocated to bands $N$ and $N+1$, with (on average) a slightly larger number of nodes from $N$. In one cluster the processor is a child, and belongs to the lower band. In the other cluster, the processor is a parent, and belongs to the higher band. The clusters are 15 Gbps interconnects.

Naturally, each node is only directly connected to a few parents and a few children. The embedding can either choose to connect more parents directly, by building a communication tree: with a branching factor of $b$, it would take at most $\log_b(CK)$ levels to allow complete interconnection between children and parents in the widest layer. Alternatively, if a node in band $n$ needs to send an event to a node in band $n+1$ that is not directly connected, it can choose to send it to any other node in band $n$ or $n+1$, and recurse.

In general, we expect to be able to embed the vrtual topology in such a way that most communication is direct. The one exception is the interconnect between the leaf band and the first level of general processors. Allocation of packets to network processors (leaf nodes) is random. Each network processor gets the $K$th packet from its channel, so it is impossible, in general, to predict which leaf node will need to speak to which parent. Assignment is random (based on the packet order on the fiber).

We expect the routing layer between the leaves and the next level to be the most complicated in the graph.

We can extend the physical fabric by adding a cluster of new processors. We want to embed logical topologies in the general FAB shape into some physical topology. Our physical topology will generally mirror the logical topology, but rather than literally being a pyramidal shaped graph, we build it out of clusters consisting of a few handful of processors. We strive to minimize the network diameter within a band (minimizing the longest path between peers), and give maximum flexibility in choosing the width and number of bands. Finally, we can calculate the expect traffic level on each logical link and we want to make sure that there is enough physical capacity to support the logical link as well as minimize latency on the larger logical links (so heavily trafficed links should always be one-hop on a local cluster).

We connect to the existing fabric by using heuristics. For example, we look for pairs of clusters that are doubly interconnected. That is, for each node in the new local cluster find a pair of already existing local clusters, $c1$ and $c2$, such that two or more nodes in $c1$ and $c2$ are connected. Break the connection between one such pair of nodes and use that to splice this node in. We may be able to precisely characterize properties of a graph that are optimal for our embedding problem.

### 5.3 Handling heavy load

Our primary method of dealing with load is reconfiguration. However, overload can still occur for two reasons. First, there is always a delay before dynamic reconfiguration can occur. Second, the choice of both a logical topology and its embedding into the physical embedding are based on expected mean traffic patterns. Our expectations may be incorrect, and there may always be sustained bursts of low probability patterns.

In such cases the only response of the system is to shed load. The way to shed load in systems such as ABIDE is to drop packets or events. We were surprised to find (although in retrospect perhaps this should have been more obvious) that the system can be very sensitive to precisely which packets get dropped. Note that classification must occur *after* aggregation (it is often impossible to classify an incomplete packet, message, or flow). This means that a packet loss early in the pipeline does not simply eliminate the packet itself, but will void any aggregate to which that packet wil subsequently belong. This is analogous to the, now well known, problem with TCP over ATM identified by Romanow and Floyd [15]. When transmitting IP packets over ABR ATM, if a single cell is dropped, it is important to drop *all* subsequent cells from that packet. Failure to do so can result in the packet loss rate multiplied by a factor of 30! (Cells allowed through will eventually be useless, and there may be 30 cells in a single packet. If the other cells are not dropped, then 29 more cells from *other* packets may need to be dropped. However, if the remaining 15-30 useless cells are preemptively dropped, then there are 15-30 other cells that may safely get through.)

A similar issue arises in ABIDE. If a single packet is to be dropped, then the rest of the pipeline must be informed, and all aggregates that this packet may belong to must be dropped, immediately, and all "hanging state" related to those aggregates can be freed up. Failure to do this can result in the detection engine showing a precipitous drop in throughput or detection rate any time there is a short term spike in incoming load.

## 6 Comparison with other research

At the time of our initial proposal, we were unaware of any other work in the area of transparent optical networks and other broadband systems that used parallel computing for scalable network surveillance. This was particularly true in the coupling of parallel hardware and WDM of ABIDE. While we ourselves have studied distributed firewalls [9], firewalling is a different problem, since the firewalls can act on a common policy without causing a large data set to be generated for analysis at a common management point.

Bellcore's Sunshine [12] ATM switch of the early 1990s provided a programmable output port controller line card which provided network embedded functionality with an Intel I960 and a substantial cell buffer, albeit at the level of framed ATM cells. The Bellcore Osiris [11] ATM adapter provided an embedded Intel I960 RISC processor to manage various cell processing tasks once ATM cells had been captured and framed with a SONET framer. These efforts illustrated the requirement for limiting processing to specialized tasks. MCI's monitoring and control system for the NSF vBNS (OC12mon) [10] uses an optical tap and relies heavily on the embedded processor in an ATM host interface to cope with traffic, mainly by selective filtering; processing performance is not as well-coupled to transport system performance as our IXP1200 leaf nodes.

The DADO tree machine architecture [7] was intended to process many production system rules in parallel. While the machine was successfully constructed and led to a high-tech startup (Fifth Generation Computer Systems), the available parallelism in OPS-5 systems did not challenge the scalability of the DADO architecture. By instead looking at the natural parallelism of WDM systems, we obtain a capability match between the machine architecture and its surveillance applications.

More recently, the ARO funded the HiDRA work at University of Colorado at Boulder, and at University of California at Santa Barbara and at Irvine. Their approach bears some striking similarity to our work with ABIDE/FAB. First, like ABIDE, they partition incoming network traffic to enable parallelization (what they call traffic scattering, slicing, and reassembly, we call routing, filtering, and aggregation). Second, they construct a web of sensors and processors to redact the incoming data stream, and analyze it for potential intrusions and events that need monitoring. Their parallel architecture is extremely similar to a static version of a single band in the logical FAB topology. (We find this an encouraging sign, that two completely independent research efforts have come up with strongly similar designs. This is circumstantial evidence that we are both at a reasonable point in the design space.)

However, there are also significant differences between our two approaches. The primary difference is where we choose to "tap" into the data network. We choose to place our supervisory channel at points of high aggregation (for example, tapping multiple wavelength optical fibers at their point of ingress to a site). HiDRA chooses to tap at all of the end-hosts and use a distributed algorithm to aggregate the data over the (possibly compromised) network. The advantage of the ABIDE approach is that no complicated aggregation is needed; all of the necessary data is locally available and is secure even if the local network is compromised. The advantage of the HiDRA approach is that each tap need only sample at speeds that are manageable by a single workstation. A second difference is in functionality: we have directly tackled the problem of dynamic reconfiguration to adapt to load, and have thought about the behavior of the system when overloaded. In order to achieve good performance (equal latency bands, maximize throughput), the network topology needs to be tuned to the set of detection rules *and* the expected pattern of network traffic. It seems clear that at any point in time, only a small fraction of the rules/sensors will actually serve to discriminate real attacks, and that the load can vary immensely. ABIDE monitors queue activity and is able to dynamically reconfigure. More importantly, even before reconfiguration, ABIDE can shed load in a way that degrades gracefully. Naive load-shedding can result in a sharp drop in performance. HiDRA can deal with this in two ways: first, by overprovisioning their sensor web, and second by assuming that load is mostly significant when under virus attack, and coupling HiDRA with a completely separate mechanism to handle viruses and denial-of-service attacks. A third difference is that ABIDE is easily scalable; we can add clusters of general purpose processors to the net and the system can dynamically utilize them. HiDRA cannot be so easily modified; however one can argue that because the HiDRA taps are co-located with end-nodes, scalability is achieved by the allocation of a new HiDRA tap with each new workstation or other end-node.

Finally, it should be noted that, differences aside, it seems that ABIDE's dynamic reconfiguration algorithms may be applicable to HiDRA, too. HiDRA is at a more mature stage in development than ABIDE/FAB. They appear to have a prototype hardware implementation, while ABIDE is still a paper design with partial simulation. Their lower level architecture may be more realizable than ABIDE's, and perhaps the best architecture is some combination of the two. Further experimentation, analysis, and comparison is needed.

## References

1. M. Maeda, "Management and Control of Transparent Optical Networks", IEEE JSAC 16(7), September 1998, pp.1008-1023.

2. J. Smith, I. Hadzic, W. Marcus, "ACTIVE Interconnects", Proc. IEEE Hot Interconnects, August 1998, pp.159-173.

3. D. J. Farber and P. Baran, "The Convergence of Computing and Telecommunications Systems", SCIENCE, Special issue on Electronics, vol. 195, pp.1166-1170, 1977.

4. J. Smith, K. Calvert, S. Murphy, H. Orman, L. Peterson, "Activating Networks", IEEE Computer 37(4), April 1999.

5. John M. Senior, Michael R. Handley and Mark S. Leeson, "Developments in Wavelength Division Multiple Access Networking", IEEE Communications Magazine 36(12), December 1998, pp. 28-38.

6. C. B. S. Traw, "Applying Architectural Parallelism in High Performance Network Subsystems," Ph.D. Thesis, University of Pennsylvania, 1995.

7. D. Miranker and S. Stolfo, "DADO: A Tree-Structured Architecture for Artificial Intelligence Computation," in Annual Reviews of Computer Science, Vol. I, 1986, Kaufmann, pp. 1-18.

8. V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Research Report LBNL-41197, Lawrence Berkeley National Laboratory, 1998.

9. S. Ioannidis, A. Keromytis, S. Bellovin, J. Smith, "Implementing a Distributed Firewall," Proc. ACM CCS, Athens, GR, 2000, pp. 190-199.

10. http://www.nlanr.net

11. B. S. Davie, "The Architecture and Implementation of a High-Speed Host Interface", IEEE JSAC, 11(2), Feb. 1993, pp. 228-239.

12. J. Giacopelli, J. Hickey, W. Marcus, W. D. Sincoskie, M. Littlewood, "Sunshine: A High-Performance Self-Routing Broadband Packet Switch Architecture", IEEE JSAC, 9(8), October 1991, pp. 1289-1298.

13. J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, E. Stoner. "State of the Practice of Intrusion Detection Technologies", CMU Software Engineering Institute Technical Report CMU/SEI-99-TR-028, January 2000.

14. W. Lee, D. Contis, J. Cabrera, D. Schimmel, A. Thomas, N. Balwalli, C. Clark, and W. Shi, "Toward High-Speed and High-Fidelity Intrusion Detection", Technical Report, College of Computing and School of Electrical and Computer Engineering Georgia Institute of Technology, Atlanta, GA

15. A. Romanow and S. Floyd, "Dynamics of TCP Traffic over ATM Networks", IEEE Journal on Selected Areas in Communications", 13(4), pp. 633-641, May, 1995.